Incremental Sparse GP Regression for Continuous-time Trajectory Estimation & Mapping

Xinyan Yan
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332, USA
xinyan.yan@cc.gatech.edu

Vadim Indelman
Department of Aerospace Engineering
Technion - Israel Institute of Technology
Haifa, 32000, Israel
vadim.indelman@technion.ac.il

Byron Boots
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332, USA
bboots@cc.gatech.edu

***Abstract*—Recent work on simultaneous trajectory estimation and mapping (STEAM) for mobile robots has found success by representing the trajectory as a Gaussian process. Gaussian processes can represent a continuous-time trajectory, elegantly handle asynchronous and sparse measurements, and allow the robot to query the trajectory to recover its estimated position at any time of interest. A major drawback of this approach is that STEAM is formulated as a *batch* estimation problem. In this paper we provide the critical extensions necessary to transform the existing batch algorithm into an extremely efficient incremental algorithm. In particular, we are able to vastly speed up the solution time through efficient variable reordering and incremental sparse updates, which we believe will greatly increase the practicality of Gaussian process methods for robot mapping and localization. Finally, we demonstrate the approach and its advantages on both synthetic and real datasets.**

I. Introduction & Related Work

Simultaneously recovering the location of a robot and a map of its environment from sensor readings is a fundamental challenge in robotics [19, 8, 1].

Well-known approaches to this problem, such as square root smoothing and mapping (SAM) [5], have focused on regression-based methods that exploit the sparse structure of the problem to efficiently compute a solution. The main weakness of the original SAM algorithm was that it was a *batch* method: all of the data must be collected before a solution can be found. For a robot traversing an environment, the inability to update an estimate of its trajectory online is a significant drawback. In response to this weakness, Kaess et al. [12] developed a critical extension to the batch SAM algorithm, incremental smoothing and mapping (iSAM), that overcomes this problem by *incrementally* computing a solution. The main drawback of iSAM, was that the approach required costly periodic batch steps for variable reordering to maintain sparsity and relinearization. This approach was extended in iSAM 2.0 [13], which employs an efficient data structure called the *Bayes tree* [14] to perform incremental variable reordering and just-in-time relinearization, thereby eliminating the bottleneck caused by batch variable reordering and relinearization. The iSAM 2.0 algorithm and its extensions are widely considered to be state-of-the-art in robot trajectory estimation and mapping.

The majority of previous approaches to trajectory estimation and mapping, including the smoothing-based SAM family of algorithms, have formulated the problem in discrete time [16, 19, 8, 1, 5, 13, 3]. However, discrete-time representations are restrictive: they are not easily extended to trajectories with irregularly spaced waypoints or asynchronously sampled measurements. A continuous-time formulation of the SAM problem where measurements constrain the trajectory at any point in time, would elegantly contend with these difficulties. Viewed from this perspective, the robot trajectory is a *function* $\boldsymbol{x}(t)$, that maps any time t to a robot state. The problem of estimating this function along with landmark locations has been dubbed *simultaneous trajectory estimation and mapping* (STEAM) [2].

Tong et al. [20] proposed a Gaussian process (GP) regression approach to solving the STEAM problem. While their approach was able to accurately model and interpolate asynchronous data to recover a trajectory and landmark estimate, it suffered from significant computational challenges: naive Gaussian process approaches to regression have notoriously high space and time complexity. Additionally, Tong et al.'s approach is a *batch* method, so updating the solution necessitates saving all of the data and completely resolving the problem. In order to combat the computational burden, Tong et al.'s approach was extended in Barfoot et al. [2] to take advantage of the sparse structure inherent in the STEAM problem. The resulting algorithm significantly speeds up solution time and can be viewed as a continuous-time analog of Dellaert's original square-root SAM algorithm [5]. Unfortunately, like SAM, Barfoot et al.'s GP-based algorithm remains a batch algorithm, which is a disadvantage for robots that need to continually update the estimate of their trajectory and environment.

In this work, we provide the critical extensions necessary to transform the existing Gaussian process-based approach to solving the STEAM problem into an extremely efficient incremental approach. Our algorithm elegantly combines the benefits of Gaussian processes and iSAM 2.0. Like the GP regression approaches to STEAM, our approach can model continuous trajectories, handle asynchronous measurements, and naturally interpolate states to speed up computation and reduce storage requirements, and, like iSAM 2.0, our approach uses a Bayes tree to efficiently calculate a *maximum a posteriori* (MAP) estimate of the GP trajectory while performing

incremental factorization, variable reordering, and just-in-time relinearization. The result is an online GP-based solution to the STEAM problem that remains computationally efficient while scaling up to large datasets.

II. Batch Trajectory Estimation & Mapping as Gaussian Process Regression

We begin by describing how the simultaneous trajectory estimation and mapping (STEAM) problem can be formulated in terms of Gaussian process regression. Following Tong et al. [20] and Barfoot et al. [2], we represent robot trajectories as functions of time t sampled from a Gaussian process:

$$\boldsymbol{x}(t) \sim \mathcal{GP}(\boldsymbol{\mu}(t), \boldsymbol{\mathcal{K}}(t,t')), \quad t_0 < t, t' \tag{1}$$

Here, $\boldsymbol{x}(t)$ is the continuous-time trajectory of the robot through state-space, represented by a Gaussian process with mean $\boldsymbol{\mu}(t)$ and covariance $\boldsymbol{\mathcal{K}}(t,t')$ functions.

We next define a finite set of measurements:

$$\boldsymbol{y}_i = \boldsymbol{h}_i(\boldsymbol{\theta}_i) + \boldsymbol{n}_i, \quad \boldsymbol{n}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_i), \quad i = 1, 2, ..., N \tag{2}$$

The measurement $\boldsymbol{y}_i$ can be any linear or nonlinear functions of a set of related variables $\boldsymbol{\theta}_i$ plus some Gaussian noise $\boldsymbol{n}_i$. The related variables for a range measurement are the robot state at the corresponding measurement time $\boldsymbol{x}(t_i)$ and the associated landmark location $\boldsymbol{\ell}_j$. We assume the total number of measurements is N, and the number of trajectory states at measurement times are M.

Based on the definition of Gaussian processes, any finite collection of robot states has a joint Gaussian distribution [18]. So the robot states at measurement times are normally distributed with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\mathcal{K}}$.

$$\begin{aligned} &\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\mathcal{K}}), \quad \boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}(t_1)^\mathsf{T} & \dots & \boldsymbol{x}(t_M)^\mathsf{T} \end{bmatrix}^\mathsf{T} \\ &\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}(t_1)^\mathsf{T} & \dots & \boldsymbol{\mu}(t_M)^\mathsf{T} \end{bmatrix}^\mathsf{T}, \quad \boldsymbol{\mathcal{K}}_{ij} = \boldsymbol{\mathcal{K}}(t_i, t_j) \end{aligned} \tag{3}$$

Note that any point along the continuous-time trajectory can be estimated from the Gaussian process model. Therefore, the trajectory does not need to be discretized and robot trajectory states do not need to be evenly spaced in time, which is an advantage of the Gaussian process approach over discrete-time approaches (e.g. Dellaert's square-root SAM [5]).

The landmarks $\boldsymbol{\ell}$ which represent the map are assumed to conform to a joint Gaussian distribution with mean $\boldsymbol{d}$ and covariance $\boldsymbol{W}$ (Eq. 4). The prior distribution of the combined state $\boldsymbol{\theta}$ that consists of robot trajectory states at measurement times and landmarks is, therefore, a joint Gaussian distribution (Eq. 5).

$$\boldsymbol{\ell} \sim \mathcal{N}(\boldsymbol{d}, \boldsymbol{W}), \quad \boldsymbol{\ell} = \begin{bmatrix} \boldsymbol{\ell}_1^\mathsf{T} & \boldsymbol{\ell}_2^\mathsf{T} & \dots & \boldsymbol{\ell}_O^\mathsf{T} \end{bmatrix}^\mathsf{T} \tag{4}$$

$$\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\eta}, \boldsymbol{\mathcal{P}}), \quad \boldsymbol{\eta} = \begin{bmatrix} \boldsymbol{\mu}^\mathsf{T} & \boldsymbol{d}^\mathsf{T} \end{bmatrix}^\mathsf{T}, \quad \boldsymbol{\mathcal{P}} = \begin{bmatrix} \boldsymbol{\mathcal{K}} & \\ & \boldsymbol{W} \end{bmatrix} \tag{5}$$

To solve the STEAM problem, given the prior distribution of the combined state and the likelihood of measurements, we compute the *maximum a posteriori* (MAP) estimate of the combined state *conditioned* on measurements via Bayes' rule:

$$\begin{aligned} \boldsymbol{\theta}^* &\triangleq \hat{\boldsymbol{\theta}}_{MAP} = \operatorname*{argmax}_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\boldsymbol{y}) = \operatorname*{argmax}_{\boldsymbol{\theta}} \frac{p(\boldsymbol{\theta})p(\boldsymbol{y}|\boldsymbol{\theta})}{p(\boldsymbol{y})} \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} p(\boldsymbol{\theta})p(\boldsymbol{y}|\boldsymbol{\theta}) = \operatorname*{argmin}_{\boldsymbol{\theta}} \left(-\log p(\boldsymbol{\theta}) - \log p(\boldsymbol{y}|\boldsymbol{\theta})\right) \\ &= \operatorname*{argmin}_{\boldsymbol{\theta}} \left(\|\boldsymbol{\theta} - \boldsymbol{\eta}\|^2_{\boldsymbol{\mathcal{P}}} + \|\boldsymbol{h}(\boldsymbol{\theta}) - \boldsymbol{y}\|^2_{\boldsymbol{R}} \right) \end{aligned} \tag{6}$$

where the norms are Mahalanobis norms defined as: $\|\boldsymbol{e}\|^2_{\boldsymbol{\Sigma}} = \boldsymbol{e}^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{e}$, and $\boldsymbol{h}(\boldsymbol{\theta})$ and $\boldsymbol{R}$ are the mean and covariance of the measurements collected, respectively:

$$\boldsymbol{h}(\boldsymbol{\theta}) = \begin{bmatrix} \boldsymbol{h}_1(\boldsymbol{\theta}_1) & \boldsymbol{h}_2(\boldsymbol{\theta}_2) & \dots & \boldsymbol{h}_N(\boldsymbol{\theta}_N) \end{bmatrix}^\mathsf{T} \tag{7}$$

$$\boldsymbol{R} = \operatorname{diag}(\boldsymbol{R}_1, \boldsymbol{R}_2, \dots, \boldsymbol{R}_N) \tag{8}$$

Because both covariance matrices $\boldsymbol{\mathcal{P}}$ and $\boldsymbol{R}$ are positive definite, the objective in Eq. 6 corresponds to a least squares problem. Consequently, if some of the measurement functions $\boldsymbol{h}_i(\cdot)$ are nonlinear, this becomes a nonlinear least squares problem, in which case iterative methods including Gauss-Newton and Levenberg-Marquardt [6] can be utilized. A linearization of a measurement function at current state estimate $\bar{\boldsymbol{\theta}}_i$ can be accomplished by a first-order Taylor expansion:

$$\boldsymbol{h}_i\left(\bar{\boldsymbol{\theta}}_i + \delta\boldsymbol{\theta}_i\right) \approx \boldsymbol{h}_i(\bar{\boldsymbol{\theta}}_i) + \left.\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{\theta}_i}\right|_{\bar{\boldsymbol{\theta}}_i} \delta\boldsymbol{\theta}_i \tag{9}$$

Combining Eq. 9 with Eq. 6, the optimal increment $\delta\boldsymbol{\theta}^*$ at the current combined state estimate $\bar{\boldsymbol{\theta}}$ is

$$\delta\boldsymbol{\theta}^* = \operatorname*{argmin}_{\delta\boldsymbol{\theta}} \left(\|\bar{\boldsymbol{\theta}} + \delta\boldsymbol{\theta} - \boldsymbol{\eta}\|^2_{\boldsymbol{\mathcal{P}}} + \|\boldsymbol{h}(\bar{\boldsymbol{\theta}}) + \boldsymbol{H}\delta\boldsymbol{\theta} - \boldsymbol{y}\|^2_{\boldsymbol{R}} \right) \tag{10}$$

Where $\boldsymbol{H}$ is the measurement Jacobian matrix:

$$\boldsymbol{H} = \operatorname{diag}(\boldsymbol{H}_1, \boldsymbol{H}_2, \dots, \boldsymbol{H}_N), \qquad \boldsymbol{H}_i = \left.\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{\theta}_i}\right|_{\bar{\boldsymbol{\theta}}_i} \tag{11}$$

To solve the linear least squares problem in Eq. 10, we take the derivative with respect to $\delta\boldsymbol{\theta}$, and set it to zero, which gives us $\delta\boldsymbol{\theta}^*$ embedded in a set of linear equations

$$\underbrace{\left(\boldsymbol{\mathcal{P}}^{-1} + \boldsymbol{H}^\mathsf{T}\boldsymbol{R}^{-1}\boldsymbol{H}\right)}_{\boldsymbol{\mathcal{I}}} \delta\boldsymbol{\theta}^* = \underbrace{\boldsymbol{\mathcal{P}}^{-1}(\boldsymbol{\eta} - \bar{\boldsymbol{\theta}}) + \boldsymbol{H}^\mathsf{T}\boldsymbol{R}^{-1}(\boldsymbol{y} - \bar{\boldsymbol{h}})}_{\boldsymbol{b}} \tag{12}$$

with covariance

$$\operatorname{cov}(\delta\boldsymbol{\theta}^*, \delta\boldsymbol{\theta}^*) = \boldsymbol{\mathcal{I}}^{-1} \tag{13}$$

The positive definite matrix $\boldsymbol{\mathcal{P}}^{-1} + \boldsymbol{H}^\mathsf{T}\boldsymbol{R}^{-1}\boldsymbol{H}$ is the *a posteriori* information matrix, which we label $\boldsymbol{\mathcal{I}}$. To solve this set of linear equations for $\delta\boldsymbol{\theta}^*$, we do not actually have to calculate the inverse $\boldsymbol{\mathcal{I}}^{-1}$. Instead, factorization-based methods can provide a fast, numerically stable solution. For example, $\delta\boldsymbol{\theta}^*$ can be found by first performing a Cholesky factorization $\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\mathsf{T} = \boldsymbol{\mathcal{I}}$, and then solving $\boldsymbol{\mathcal{L}}\boldsymbol{d} = \boldsymbol{b}$ and $\boldsymbol{\mathcal{L}}^\mathsf{T}\delta\boldsymbol{\theta}^* = \boldsymbol{d}$ by back substitution. At each iteration we perform a *batch* state estimation update $\bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + \delta\boldsymbol{\theta}^*$ and repeat the process until convergence.

If $\boldsymbol{\mathcal{I}}$ is dense, the time complexity of a Cholesky factorization and back substitution are $O(n^3)$ and $O(n^2)$ respectively,

where $\boldsymbol{\mathcal{I}} \in \mathbb{R}^{n \times n}$ [9]. However, if $\boldsymbol{\mathcal{I}}$ has sparse structure, then the solution can be found much faster. For example, for a narrowly banded matrix, the computation time is $O(n)$ instead of $O(n^3)$ [9]. Fortunately, we can guarantee sparsity for the STEAM problem (see Section II-B below).

A. State Interpolation

An advantage of the Gaussian process representation of the robot trajectory is that any trajectory state can be interpolated from other states by computing the posterior mean [20]:

$$\bar{\boldsymbol{x}}(t) = \boldsymbol{\mu}(t) + \boldsymbol{\mathcal{K}}(t)\boldsymbol{\mathcal{K}}^{-1}(\bar{\boldsymbol{x}} - \boldsymbol{\mu}), \tag{14}$$

with

$$\begin{aligned} \bar{\boldsymbol{x}} &= \begin{bmatrix} \bar{\boldsymbol{x}}(t_1) & \dots & \bar{\boldsymbol{x}}(t_M) \end{bmatrix}^\mathsf{T} \quad \text{and} \\ \boldsymbol{\mathcal{K}}(t) &= \begin{bmatrix} \boldsymbol{\mathcal{K}}(t, t_1) & \dots & \boldsymbol{\mathcal{K}}(t, t_M) \end{bmatrix}. \end{aligned} \tag{15}$$

By utilizing interpolation, we can reduce the number of robot trajectory states that we need to estimate in the optimization procedure [20]. For simplicity, assume $\boldsymbol{\theta}_i$, the set of the related variables of the ith measurement according to the model (Eq. 2), is $\boldsymbol{x}(t_j)$. Then, after interpolation, Eq. 9 becomes:

$$\begin{aligned} &\boldsymbol{h}_i\left(\bar{\boldsymbol{\theta}}_i + \delta\boldsymbol{\theta}_i\right) = \boldsymbol{h}_i\left(\bar{\boldsymbol{x}}(t_j) + \delta\boldsymbol{x}(t_j)\right) \\ &\approx \boldsymbol{h}_i(\bar{\boldsymbol{x}}(t_j)) + \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{x}(t_j)} \cdot \frac{\partial \boldsymbol{x}(t_j)}{\partial \boldsymbol{x}}\bigg|_{\bar{\boldsymbol{x}}} \delta\boldsymbol{x} \\ &= \boldsymbol{h}_i\big(\boldsymbol{\mu}(t_j) + \boldsymbol{\mathcal{K}}(t_j)\boldsymbol{\mathcal{K}}^{-1}(\bar{\boldsymbol{x}} - \boldsymbol{\mu})\big) + \boldsymbol{H}_i\boldsymbol{\mathcal{K}}(t_j)\boldsymbol{\mathcal{K}}^{-1}\delta\boldsymbol{x} \end{aligned} \tag{16}$$

By employing Eq. 16 during optimization, we can make use of measurement i without explicitly estimating the trajectory states that it relates to. We exploit this advantage to greatly speed up the solution to the STEAM problem in practice (Section V).

B. Sparse Gaussian Process Regression

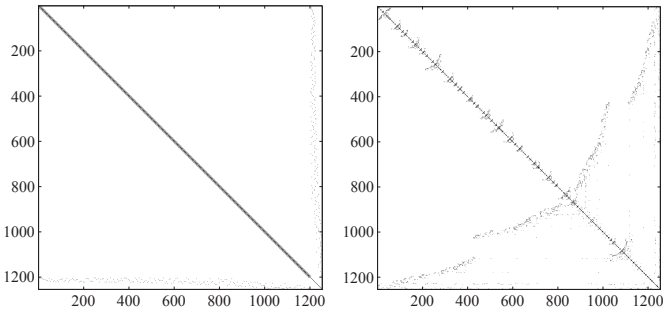
The efficiency of the Gaussian Process Gauss-Newton algorithm presented in Section II is heavily dependent on the choice of kernel. It is well-known that if the information matrix $\boldsymbol{\mathcal{I}}$ is sparse, then it is possible to very efficiently compute the solution to Eq. 12 [5]. Barfoot et al. suggest a kernel matrix with a sparse inverse that is well-suited to the simultaneous trajectory estimation and mapping problem [2]. In particular, Barfoot et al. show that $\boldsymbol{\mathcal{K}}^{-1}$ is exactly block-tridiagonal when the GP is assumed to be generated by linear, time-varying (LTV) stochastic differential equation (SDE) which we describe here:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}(t)\boldsymbol{x}(t) + \boldsymbol{v}(t) + \boldsymbol{F}(t)\boldsymbol{w}(t), \tag{17}$$

$$\boldsymbol{w}(t) \sim \mathcal{GP}(\boldsymbol{0},\, \boldsymbol{Q}_c\delta(t - t')) \quad t_0 < t, t' \tag{18}$$

where $\boldsymbol{x}(t)$ is trajectory, $\boldsymbol{v}(t)$ is known exogenous input, $\boldsymbol{w}(t)$ is process noise, and $\boldsymbol{F}(t)$ is time-varying system matrix. The process noise $\boldsymbol{w}(t)$ is modeled by a Gaussian process, and $\delta(\cdot)$ is the *Dirac delta function*. (See [2] for details). We consider a specific case of this model in the experimental results in Section V-A.

Assuming the GP is generated by Eq. 17, the measurements are landmark and odometry measurements, and the variables are ordered in XL ordering[1], the sparse information matrix becomes



(a) XL ordering[1] (b) SYMAMD ordering[2]

Fig. 1: Sparse information matrices. The information matrix $\boldsymbol{\mathcal{I}}$ with XL ordering[1] and SYMAMD ordering[2]. Both sparse matrices have the same number of non-zero elements, yet the second matrix can be factored much more efficiently due to the heuristic ordering of the matrix columns. (See Table I). For illustration, only 200 trajectory states are shown here.

$$\boldsymbol{\mathcal{I}} = \begin{bmatrix} \boldsymbol{\mathcal{I}}_{xx} & \boldsymbol{\mathcal{I}}_{x\ell} \\ \boldsymbol{\mathcal{I}}_{x\ell}^\mathsf{T} & \boldsymbol{\mathcal{I}}_{\ell\ell} \end{bmatrix} \tag{19}$$

where $\boldsymbol{\mathcal{I}}_{xx}$ is block-tridiagonal and $\boldsymbol{\mathcal{I}}_{\ell\ell}$ is block-diagonal. $\boldsymbol{\mathcal{I}}_{x\ell}$'s density depends on the frequency of landmark measurements, and how they are taken. See Fig. 1a for an example.

When the GP is generated by LTV SDE, Barfoot et al. prove that $\boldsymbol{\mathcal{K}}(t)\boldsymbol{\mathcal{K}}^{-1}$ in Eq. 14 has a specific sparsity pattern, only two column blocks that correspond to trajectory states at t_i and t_{i+1}, where $t_i < t < t_{i+1}$, are nonzero. In other words, $\bar{\boldsymbol{x}}(t)$ is an affine function of only two nearby states $\bar{\boldsymbol{x}}(t_i)$ and $\bar{\boldsymbol{x}}(t_{i+1})$:

$$\begin{aligned} \bar{\boldsymbol{x}}(t) =& \boldsymbol{\mu}(t) + \boldsymbol{\Lambda}(t)\left(\bar{\boldsymbol{x}}(t_i) - \boldsymbol{\mu}(t_i)\right) + \boldsymbol{\Psi}(t)\left(\bar{\boldsymbol{x}}(t_{i+1}) - \boldsymbol{\mu}(t_{i+1})\right), \\ & t_i < t < t_{i+1} \end{aligned} \tag{20}$$

Thus, it only takes $O(1)$ time to query any $\bar{\boldsymbol{x}}(t)$ using Eq. 20. Moreover, because interpolation of a state is only determined by the two nearby states, measurement interpolation in Eq. 16 can be significantly simplified.

III. Batch GP-Regression with Variable Reordering

Previous work on batch continuous-time trajectory estimation as sparse Gaussian process regression [20, 2] assumes that the information matrix $\boldsymbol{\mathcal{I}}$ is sparse (Eq. 19) and applies standard block elimination to factor and solve Eq. 12. Despite the sparsity of $\boldsymbol{\mathcal{I}}$, for large numbers of landmarks this process can be very inefficient. Inspired by square root SAM [5], which uses variable reordering for efficient Cholesky factorization in a discrete-time context, we show that factorization-time can be dramatically improved by matrix column reordering in the sparse Gaussian process context as well.

[1] XL ordering is an ordering where process variables come before landmarks variables.

TABLE I: Cost of Cholesky factorization with different ordering methods including ordering time

	XL[1]	SYMAMD	Block SYMAMD
nnz [3]	1817499	192285	176105
time (sec)	0.967720	0.027402	0.017500

It is reasonable to base our approach on SAM because the information matrix and factor graph of the sparse GP [2] has structure similar to the SAM formulations of the problem [5, 12], and the intuitions from previous discrete-time approaches apply here. If the Cholesky decompositions are performed naively, fill-in can occur, where entries that are zero in the information matrix become non-zero in the Cholesky factor. This occurs because the Cholesky factor of a sparse matrix is guaranteed to be sparse for some variable orderings, but not all variable orderings [17]. Therefore, we want to find a good variable ordering so that the Cholesky factor is sparse.

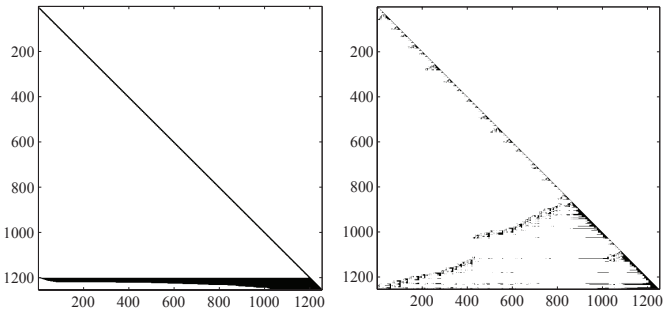
Although finding the optimal ordering for a symmetric positive definite matrix is NP-complete [21], good heuristics do exist. One such heuristic is symmetric approximate minimum degree permutation (SYMAMD)[2] [4]. To demonstrate the benefits of variable reordering, we constructed a synthetic example and compared different approaches. The example, which is explained in detail in Section V-A, consists of 1,500 time steps with trajectory states, $\boldsymbol{x}(t_i) = [\ \boldsymbol{p}(t_i)\quad \dot{\boldsymbol{p}}(t_i)\ ]^\intercal$, $\boldsymbol{p}(t_i) = [\ x(t_i)\quad y(t_i)\quad \theta(t_i)\ ]^\intercal$, and with odometry and range measurements. The total number of landmarks is 298. The structure of the information matrix $\boldsymbol{\mathcal{I}}$ and Cholesky factor $\boldsymbol{\mathcal{L}}$, with and without variable reordering, are compared in Fig. 1 and Fig. 2. Although variable reordering does not change the sparsity of the information matrix $\boldsymbol{\mathcal{I}}$ (Fig. 1), it dramatically increases the sparsity of the Cholesky factor $\boldsymbol{\mathcal{L}}$ (Fig. 2). Table I demonstrates this clear benefit of reordering. The Cholesky factor after SYMAMD ordering contains 10.6% non-zeroes of XL ordering 1, and takes 2.83% of the time, which are significant improvements in both time and space complexity.

We also experimented with block SYMAMD [5], which exploits domain knowledge to group together variables belonging to a particular trajectory state $\boldsymbol{x}(t_i)$ or landmark location $\boldsymbol{\ell}_j$ before performing SYMAMD and empirically further improves performance.

It is straightforward to incorporate variable reordering methods like SYMAMD and block SYMAMD into the batch GP-Regression algorithm from Section II. Given a new batch of data, directly update the sparse information matrix $\boldsymbol{\mathcal{I}}$, reorder the variables with (block) SYMAMD, and then recompute the Cholesky factor $\boldsymbol{\mathcal{L}}$ on the way to solving for $\delta\boldsymbol{\theta}$ in Eq. 12.

In most STEAM problems, we are interested in estimating the robot's trajectory *as it traverses the environment*. In Alg. 1, we accomplish this by repeatedly executing the batch

[2]SYMAMD is a variant of column approximate minimum degree ordering (COLAMD) [4] on positive definite matrix.



(a) XL ordering[1] (b) SYMAMD ordering[2]

Fig. 2: The Cholesky factors $\boldsymbol{\mathcal{L}}$ of $\boldsymbol{\mathcal{I}}$. In (a), $\boldsymbol{\mathcal{L}}$ is computed with XL ordering[1], which exhibits fill-in. When computed with SYMAMD ordering in (b), $\boldsymbol{\mathcal{L}}$ is more sparse. For illustration, only 200 states are shown here.

algorithm with variable reordering. Although this approach seems like it should be very costly, with variable reordering this method it is actually quite efficient. Building and factoring the sparse information matrix is much faster than the linearization step required for a single iteration of the Gauss-Newton algorithm. Since the computational bottleneck is not the Cholesky decomposition, but rather the relinearization of the measurement model, we suggest only periodic Gauss-Newton iterations.

Algorithm 1 Periodic Batch Sparse GP Regression

while collecting data **do**

1. Get measurement results belonging to this period, $\boldsymbol{y} \leftarrow [\boldsymbol{y}, \boldsymbol{y}_{new}]^\intercal$
2. Initial guess for the newly encountered states, $\bar{\boldsymbol{\theta}} \leftarrow [\bar{\boldsymbol{\theta}}, \bar{\boldsymbol{\theta}}_{new}]^\intercal$
3. Build the measurement Jacobian $\boldsymbol{H}$, and then $\boldsymbol{\mathcal{I}}$ and $\boldsymbol{b}$ required in Eq. 12
4. Find an ordering p for $\boldsymbol{\mathcal{I}}$, and reorder $\boldsymbol{\mathcal{I}}_p \xleftarrow{p} \boldsymbol{\mathcal{I}}$, $\boldsymbol{b}_p \xleftarrow{p} \boldsymbol{b}$
5. Solve $\boldsymbol{\mathcal{I}}_p \delta\boldsymbol{\theta}^*_p = \boldsymbol{b}_p$ using Cholesky factorization
6. Recover the solution $\delta\boldsymbol{\theta}^* \xleftarrow{r} \delta\boldsymbol{\theta}^*_p$ by inverse ordering $r = p^{-1}$
7. Update estimate $\bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + \delta\boldsymbol{\theta}^*$

end while

IV. Bayes Tree for Fast Incremental Updates to Sparse GP Regression

Despite the efficiency of periodic batch updates, Alg. 1 is still repeatedly executing a batch algorithm that requires reordering and refactoring $\boldsymbol{\mathcal{I}}$, and periodically relinearizing the measurement function for all of the estimated states each time new data is collected. Here we provide the extensions necessary to avoid these costly steps and turn the naive batch algorithm into an efficient, truly incremental, algorithm. The key idea is to perform just-in-time relinearization and to efficiently *update* an existing sparse factorization instead of re-calculating one from scratch.

[3]The number of non-zero elements.

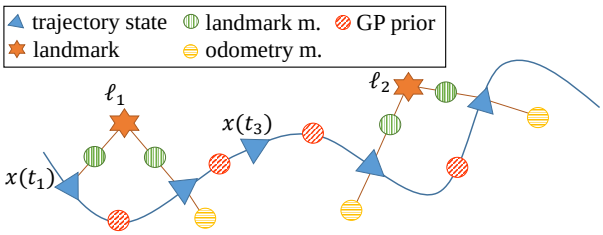


Fig. 3: A simple factor graph that includes landmark measurements, odometry measurements, and Gaussian process priors.

A. The Bayes Tree Data Structure

We base our approach on iSAM 2.0 proposed by Kaess et al. [13], which was designed to efficiently solve a nonlinear estimation problem in an incremental and real-time manner by directly operating on the factor graph representation of the SAM problem. The core technology behind iSAM 2.0 is the *Bayes tree* data structure which allows for incremental variable reordering and fluid relinearization [14]. Demonstrated by Kaess et al. [13], Bayes tree provides dramatic speedup compared to batch method, with negligible loss in accuracy. We apply the same data structure to sparse Gaussian process regression in the context of the STEAM problem, thereby eliminating the need for periodic batch computation.

To understand how the Bayes tree is used, it is helpful to understand how the GP estimation problem can be represented as a factor graph [15]. Formally, a factor graph is a bipartite graph $G = (\mathcal{F}, \boldsymbol{\theta}, \mathcal{E})$, where $\mathcal{F}$ is the set of factor nodes that encodes all probabilistic constraints on variables, including landmark measurements, odometry measurements, and smoothing priors, $\boldsymbol{\theta}$ is the set of variable nodes to estimate, and $\mathcal{E}$ is the set of edges that connect factors nodes with variable nodes. The joint probability of variables to estimate is factored as

$$f(\boldsymbol{\theta}) = \prod_i f_i(\boldsymbol{\theta}_i) \tag{21}$$

where $f_i \in \mathcal{F}$ is one of the factors, and $\boldsymbol{\theta}_i$ is the set of variables directly connected to f_i. The estimation problem is to find $\boldsymbol{\theta}^*$ that maximizes Eq. 21. As stated in Section II-B, Barfoot et al. prove that when GPs are generated by LTV SDE, $\boldsymbol{\mathcal{K}}^{-1}$ is block-tridiagonal [2]. They also state that the factors resulted from the Gaussian process representation of the trajectory, or the Gaussian process prior factors, only connect consecutive pairs of states. This leads to a sparse Factor graph (Fig. 3).

The factor graph can be converted to a Bayes net by a *bipartite elimination game* [11]. The procedure is equivalent to converting Eq. 10 to least squares form and computing the square-root information matrix $\boldsymbol{\mathcal{L}}$ via an incomplete Cholesky factorization. To facilitate marginalization and optimization, a *Bayes tree*, which groups several variables together based on their dependence, is constructed from the Bayes net [14]. From a linear algebra perspective, the Bayes tree captures the structure of the Cholesky factor $\boldsymbol{\mathcal{L}}$ of $\boldsymbol{\mathcal{I}}$, and the sequence of back substitutions that can be performed.

When we add a new measurement, add a prior for a new variable, or relinearize a previous measurement, $\boldsymbol{\mathcal{L}}$ will change accordingly. However, all modifications to the factor graph only have *local* effects on $\boldsymbol{\mathcal{L}}$. Exploiting this observation is the foundation for efficient incremental updates. Since the nodes of Bayes tree encode conditional probability distributions which directly correspond to rows in $\boldsymbol{\mathcal{L}}$, the structure of the tree can be leveraged to efficiently update the factor $\boldsymbol{\mathcal{L}}$ [14].

The nodes $\boldsymbol{\theta}_{nf}$ containing variables involved in new factors, or nodes $\boldsymbol{\theta}_{lin}$ whose linear step is larger than a predetermined threshold, are identified. [4]

Only these nodes and their ascendants in the Bayes tree are then updated. When a sub-tree is updated, variables in the sub-tree are reordered by constrained COLAMD [4] to ensure sparsity and heuristically maximize the locality of future updates. Finally, $\delta\boldsymbol{\theta}^*$ is computed from tree root to leaves. Propagation stops when updates to the conditioning variables are below a predetermined threshold.

Algorithm 2 summarizes incremental Gaussian process regression with the Bayes tree data structure in detail. The algorithm also incorporates state interpolation, described in the next section.

B. Faster Updates Through Interpolation

To further reduce computation time, we take advantage of Gaussian process state interpolation (as suggested by Tong et al. [20]) within our incremental algorithm. This allows us to reduce the total number of estimated states, while still using all of the measurements, including those that involve interpolated states. By only estimating a small fraction of the states along the trajectory, we realize a significant speedup relative to a naive application of the Bayes tree (see Section V). This is an advantage of GP-based methods with respect to discrete-time methods like iSAM 2.0.

Algorithm 2 describes how interpolation is used within our incremental algorithm: First, when a measurement related to a missing state is received, the variables necessary to interpolate the state, as well as the corresponding cliques in Bayes tree that should be removed or updated, are identified. Since the sparse GP has a LTV SDE prior, each interpolated state is only a function of two nearby states (see Eq. 20). These nearby states are therefore included into the set of variables $\boldsymbol{\theta}_{nf}$ related to the new factor (line 1). In the case that the GP relies on a different kernel matrix, the corresponding states used for interpolation can be determined from Eq. 14. Second, linearization of factors that involve missing states (line 3) is performed by incorporating state interpolation via Eq. 16.

V. Experimental Results

We evaluate the performance of our incremental sparse GP regression algorithm to solving the STEAM problem on synthetic and real-data experiments and compare our approach to the state-of-the-art. In particular, we evaluate how variable reordering can dramatically speed up the batch solution to the sparse GP regression problem, and how, by utilizing the Bayes

[4]The reader is referred to [13] for additional details regarding this just-in-time relinearization.

Algorithm 2 Updating Sparse GP Regression by Bayes Tree

while collecting data **do**

1. Get new measurement results, store new factors $\mathcal{F}_{new}$ and identify related variables $\boldsymbol{\theta}_{nf} = \bigcup \boldsymbol{\theta}_i$, $f_i \in \mathcal{F}_{new}$. If the state $\boldsymbol{x}(t_i) \in \boldsymbol{\theta}_{nf}$ is missing, then it is replaced by variables used in interpolation (Eq. 20); If $\boldsymbol{x}(t_i) \in \boldsymbol{\theta}_{nf}$ is a new state to estimate, the previous state to estimate is added to $\boldsymbol{\theta}_{nf}$, and a Gaussian process prior factor is stored.
2. For each affected variable in $\boldsymbol{\theta}_{aff} = \boldsymbol{\theta}_{lin} \cup \boldsymbol{\theta}_{nf}$, remove the corresponding clique and ascendants up to the root of Bayes tree.
3. Relinearize the factors required to recreate the removed part. Use interpolation when linearizing factors involving missing states (Eq. 16)
4. Add cached marginal factors from orphaned sub-trees of removed cliques and create a factor graph
5. Eliminate the factor graph by a new variable ordering, create a Bayes tree, and attach back orphaned sub-trees
6. Partially update estimate from root to leaves and stop walking down a branch when the updates to variables that the child clique is conditioned on are not significant enough
7. Collect variables involved in the measurement factors $\mathcal{F}_{lin}$ where previous linearization point is far from current estimate, $\boldsymbol{\theta}_{lin} = \bigcup \boldsymbol{\theta}_i$, $f_i \in \mathcal{F}_{lin}$

end while

tree and interpolation for incremental updates, our algorithm can yield even greater gains in the online trajectory estimation scenario. We compare:

- **PB**: Periodic batch (described in Section II). This is the state-of-the-art algorithm presented in Barfoot et al. [2] (XL variable ordering), which is periodically executed as data is received.
- **PBVR**: Periodic batch with variable reordering (described in Section III).
- **BTGP**: The proposed approach - Bayes tree with Gaussian process prior factors (described in Section IV).

If the GP is only used to estimate the state at measurement times, the proposed approach offers little beyond a reinterpretation of the standard discrete-time iSAM 2.0 algorithm. Therefore, we also compare our GP-based algorithm, which leverages interpolation, to the standard Bayes tree approach used in iSAM 2.0. We show that by interpolating large fractions of the trajectory during optimization, the GP allows us to realize significant performance gains over iSAM 2.0 with minimal loss in accuracy. For these experiments we compare:

- **without interpolation**: BTGP without interpolation at a series of lower temporal resolutions. Without interpolation BTGP is algorithmically identical to iSAM 2.0. Measurements between two estimated states are simply ignored.
- **with interpolation**: BTGP with interpolation at a series of lower resolutions. In contrast to the above case, measurements between estimated states are fully utilized by interpolating missing states at measurement times (described in Section IV-B).
- **finest estimate**: The baseline. BTGP at the finest resolution, estimating all states at measurement times. When measurements are synchronous with evenly-spaced waypoints and no interpolation is used, BTGP is identical to iSAM 2.0 applied to the full dataset.

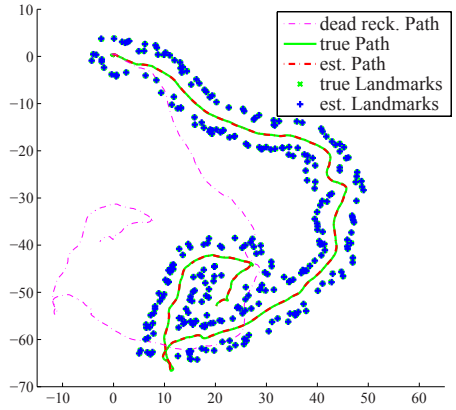


Fig. 4: Synthetic dataset: Ground truth and state estimates are shown. Lines between trajectory points and landmarks indicate range measurements. State estimates obtained from BTGP approach are very close to ground truth.

All algorithms are implemented with the same C++ libray, GTSAM 3.2,[5] to make the comparison fair and meaningful. Evaluation is performed on three datasets summarized in Table II. We first evaluate performance in a synthetic dataset (Section V-A), analyzing estimation errors with respect to ground truth data. Results using real-world datasets are then presented in Sections V-B and V-C.

A. Synthetic SLAM Exploration Task

This dataset consists of an exploration task with 1,500 time steps. Each time step contains a trajectory state $\boldsymbol{x}(t_i) = [\ \boldsymbol{p}(t_i) \quad \dot{\boldsymbol{p}}(t_i)\ ]^\intercal$, $\boldsymbol{p}(t_i) = [\ x(t_i) \quad y(t_i) \quad \theta(t_i)\ ]^\intercal$, an odometry measurement, and a range measurement related to a nearby landmark. The total number of landmarks is 298. The trajectory is randomly sampled from a Gaussian process generated from white noise acceleration $\ddot{\boldsymbol{p}}(t) = \boldsymbol{w}(t)$, i.e. constant velocity, and with zero mean.

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{F}\boldsymbol{w}(t) \tag{22}$$

where

$$\boldsymbol{x}(t) = \begin{bmatrix} \boldsymbol{p}(t) \\ \dot{\boldsymbol{p}}(t) \end{bmatrix}, \quad \boldsymbol{p}(t) = \begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \end{bmatrix}, \quad \boldsymbol{A} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix},$$
$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{I} \end{bmatrix}, \quad \boldsymbol{w}(t) \sim \mathcal{GP}(\boldsymbol{0}, \boldsymbol{Q}_c \delta(t - t')) \tag{23}$$

Note that velocity $\dot{\boldsymbol{p}}(t)$ must to be included in trajectory state to represent the motion in LTV SDE form [2].

The odometry and range measurements with Gaussian noise are specified in Eq. 24 and Eq. 25 respectively.

$$\boldsymbol{y}_{io} = \begin{bmatrix} \cos\theta(t_i) \cdot \dot{x}(t_i) + \sin\theta(t_i) \cdot \dot{y}(t_i) \\ \dot{\theta}(t_i) \end{bmatrix} + \boldsymbol{n}_o \tag{24}$$

$$y_{ir} = \left\| \begin{bmatrix} x(t_i) & y(t_i) \end{bmatrix}^\intercal - \boldsymbol{\ell}_j \right\|_2 + n_r \tag{25}$$

[5] https://collab.cc.gatech.edu/borg/gtsam/

TABLE II: Summary of experimental datasets

	# time steps	# odo. m.	# landmark m.	# landmarks	travel dist.(km)
Synthetic	1,500	1,500	1,500	298	0.2
Auto. Mower	9,658	9,658	3,529	4	1.9
Victoria Park	6,969	6,969	3,640	151	3.5

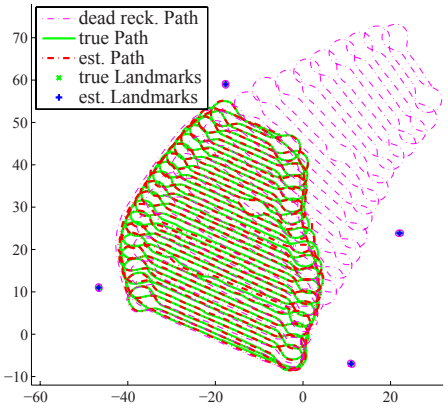


Fig. 6: The Autonomous Lawnmower dataset: Ground truth and state estimates are shown. The range measurements are sparse, noisy, and asynchronous. Ground truth and state estimates obtained from BTGP are very close.

where $\boldsymbol{y}_{io}$ consists of the robot-oriented velocity and heading angle velocity with Gaussian noise, and y_{ir} is the distance between the robot and a specific landmark $\boldsymbol{\ell}_j$ at t_i with Gaussian noise.

We compare the computation time of the three approaches (PB, PBVR and BTGP) in Fig. 5. The incremental Gaussian process regression (BTGP) offers significant improvements in computation time compared to the batch approaches (PBVR and PB).

We also demonstrate that BTGP can further increase speed over a naive application of the Bayes tree (e.g. iSAM 2.0) without sacrificing much accuracy by leveraging interpolation. To illustrate the trade-off between the accuracy and time efficiency due to interpolation, we plot RMSE of distance errors and the total computation time by varying the time step difference (the rate of interpolation) between estimated states.

B. The Autonomous Lawnmower

The second experiment evaluates our approach on real data from a freely available range-only SLAM dataset collected from an autonomous lawn-mowing robot [7]. The "Plaza" dataset consists of odometer data and range data to stationary landmarks collected via time-of-flight radio nodes. (Additional details on the experimental setup can be found in [7].) Ground truth paths are computed from GPS readings and have 2cm accuracy according to [7].

The environment, including the locations of the landmarks and the ground truth paths, are shown in Fig. 6. The robot travelled 1.9km, occupied 9,658 poses, and received 3,529 range measurements, while following a typical path generated during mowing. The dataset has sparse range measurements, but contains odometry measurements at each time step. The results of incremental BTGP are shown in Fig. 6 and demonstrate that we are able to estimate the robot's trajectory and map with a very high degree of accuracy.

As in Section V-A, performance of three approaches – periodic batch relinearization (PB), periodic batch relinearization with variable reordering (PBVR) and incremental Bayes tree (BTGP) are compared in Fig. 7.

In this dataset, the number of landmarks is 4, which is extremely small relative to the number of trajectory states, so there is no performance gain from reordering. However, the Bayes tree-based approach dramatically outperforms the other two approaches. As the problem size increases, there is negligible increase in computation time, even for close to 10,000 trajectory states.

C. Victoria Park

The third experiment evaluates our approach on the Victoria Park dataset [10], which consists of range-bearing measurements to landmarks, and speed and steering odometry measurements. The data was collected from a vehicle equipped with a laser sensor driving through the Sydney's Victoria Park. The environment contains a high number of trees as landmarks. The vehicle travelled ~ 3.5 km in 26 minutes. After repeated measurements, taken when the vehicle is stationary, are dropped, the dataset consists of 6,969 time steps and 3,640 range-bearing measurements relative to 151 landmarks. The bearing measurement is specified in Eq. 26, as the relative angle from vehicle heading to the landmark direction with Gaussian noise where $\begin{bmatrix} x_j & y_j \end{bmatrix}^\intercal$ is location of landmark j.

$$y_{ib} = \text{atan2}\left(y_j - y(t_i), x_j - x(t_i)\right) - \theta(t_i) + n_{ib} \qquad (26)$$

The results, shown in Figure 8, further demonstrate the advantages of BTGP. As seen from the upper right plot, variable reordering drastically reduces computation time when used within batch optimization (PBVR), and even further in the incremental algorithm (BTGP).

VI. Conclusion

We have introduced an incremental sparse Gaussian process regression algorithm for computing the solution to the continuous-time simultaneous trajectory estimation and mapping (STEAM) problem. The proposed algorithm elegantly combines the benefits of Gaussian process-based approaches to STEAM while simultaneously employing state-of-the-art innovations from incremental discrete-time algorithms for

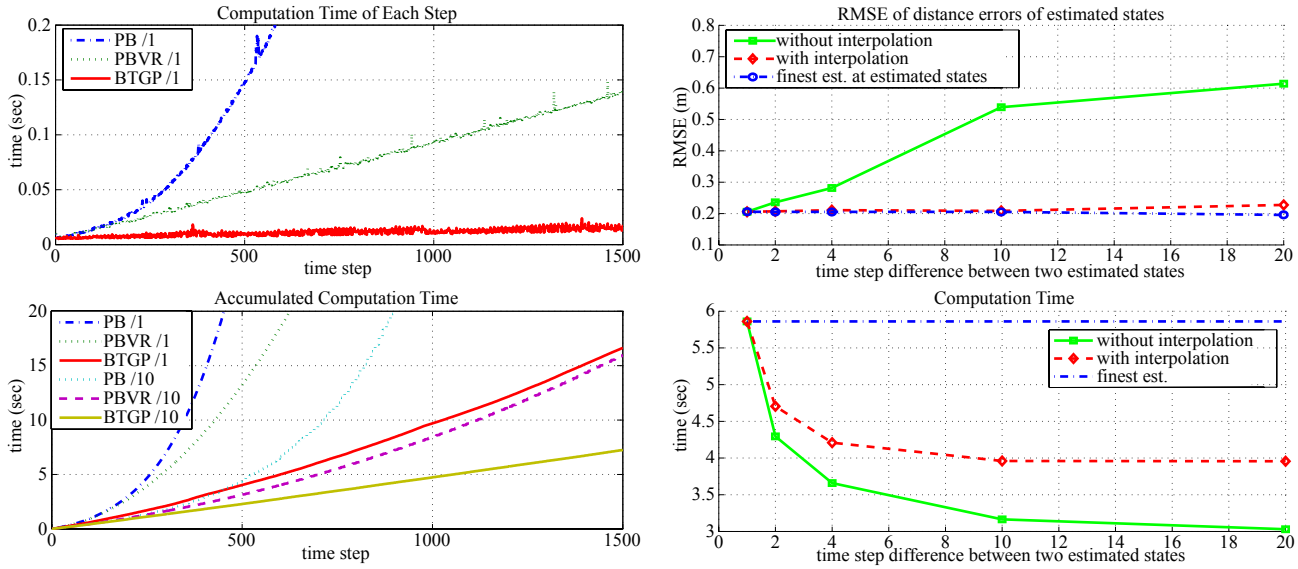


Fig. 5: Synthetic dataset: **(Left Column)** Comparison of the computation time of three approaches PB, PBVR, and BTGP. The modifiers /1 and /10 indicate frequency of state updates. For example: *BTGP/1* updates the estimate after 1 new range measurement using BTGP. Likewise *BTGP/10* updates the estimate after 10 new range measurements using BTGP. For fair comparison no interpolation is used by BTGP. Due to the large number of landmarks, 298, compared to the number of trajectory states, variable reordering dramatically improves the performance. **(Right Column)** Trade-off between computation time and accuracy if BTGP makes use of interpolation. The y-axis measures the RMSE of distance errors of the estimated trajectory states and total computation time with increasing amounts of interpolation. The x-axis measures the time step difference between two estimated (non-interpolated) states. "Without interpolation" means that the number of states is reduced, but measurements taken at the missing states are ignored. This is equivalent to running iSAM 2.0 to find a trajectory with coarse discretization. "With interpolation" is the BTGP algorithm that interpolates missing states while incorporating odometry measurements at the interpolated states. "Finest estimate" is the baseline which measures RMSE and computation time if the number of states is not reduced. This is exactly equivalent to iSAM 2.0 run on the full measurement and odometry data. The results indicate that interpolating $\sim 90\%$ of the states (i.e. estimating only $\sim 10\%$ of the states) while running BTGP can result in a 33% reduction in computation time over iSAM 2.0 without sacrificing accuracy.

smoothing and mapping. Our empirical results show that by parameterizing trajectories with a small number of states and utilizing Gaussian process interpolation, our algorithm can realize large gains in speed over iSAM 2.0 with very little loss in accuracy (e.g. reducing computation time by 68% while increasing RMSE by only 8cm on the Autonomous Lawnmower Dataset) .

REFERENCES

[1] T. Bailey and H. Durrant-Whyte. Simultaneous localisation and mapping (SLAM): Part II state of the art. *Robotics and Automation Magazine*, 13(3):108–117, 2006.

[2] Tim Barfoot, Chi Hay Tong, and Simo Sarkka. Batch continuous-time trajectory estimation as exactly sparse gaussian process regression. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.

[3] Byron Boots and Geoffrey J. Gordon. A spectral learning approach to range-only SLAM. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

[4] Timothy A. Davis, John R. Gilbert, Stefan I. Larimore, and Esmond G. Ng. Algorithm 836: Colamd, a column approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):377–380, September 2004. ISSN 0098-3500.

[5] Frank Dellaert and Michael Kaess. Square root sam: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Reasearch*, 25:2006, 2006.

[6] J. E. Dennis, Jr. and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations (Classics in Applied Mathematics, 16)*. Soc for Industrial & Applied Math, 1996. ISBN 0898713641.

[7] Joseph Djugash. *Geolocation with Range: Robustness, Efficiency and Scalability*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, November 2010.

[8] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localisation and mapping (slam): Part i the essential algorithms. *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, 2:2006, 2006.

[9] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press,

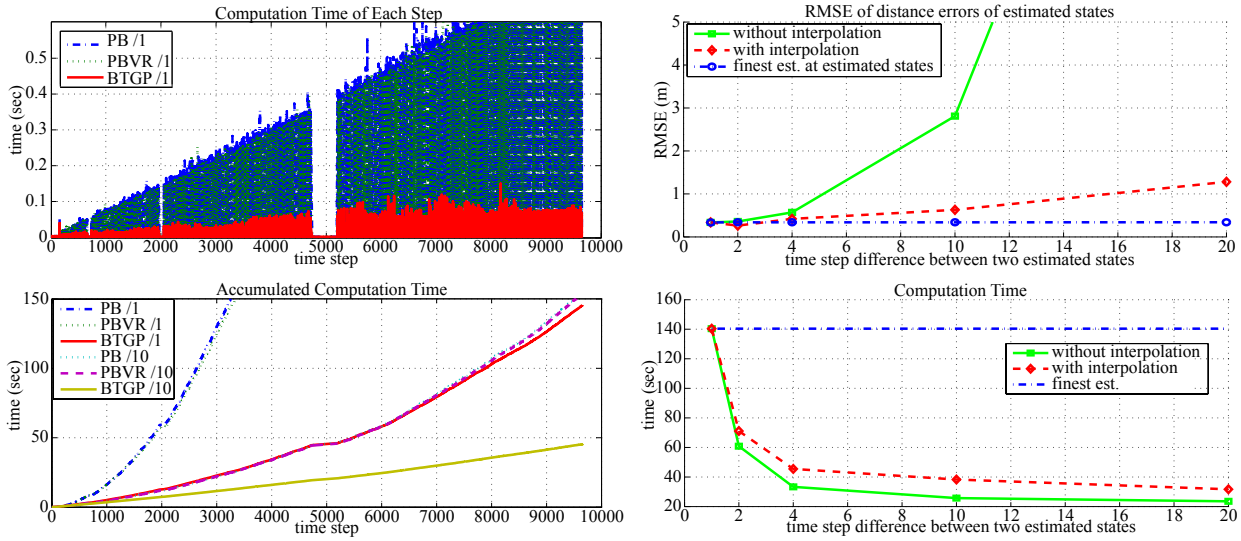


Fig. 7: Autonomous Lawnmower dataset: (**Left Column**) Comparison of the computation time of three approaches PB, PBVR, and BTGP. As in Figure 5, the modifiers /1 and /10 indicate frequency of state updates. For fair comparison no interpolation is used by BTGP in these experiments. Updates are very fast (close to zero time) when there are no range measurements. This dataset has no range measurements for a long, sustained stretch around the 5000th tilmestep, which accounts for the 'gap' in the upper left-hand graph. Due to the low number of landmarks, variable reordering does not help (PB and PBVR take roughly the same amount of time). The incremental BTGP approach dramatically reduces computation time. (**Right Column**) Trade-off between computation time and accuracy if BTGP makes use of interpolation. The y-axis measures the RMSE of distance errors of the estimated trajectory states and total computation time with increasing amounts of interpolation. The x-axis measures the time step difference between two estimated (non-interpolated) states. "Without interpolation" means that the number of states are reduced, but measurements taken at the missing states are ignored. "With interpolation" is the standard BTGP algorithm that interpolates missing states. "Finest estimate" is the baseline which measures RMSE and computation time if no states are interpolated, which is exactly equivalent to running iSAM 2.0 on the full set of measurements and odometry. The results indicate that interpolating $\sim 80\%$ of the states within BTGP results in only an 8cm increase in RSME while reducing the overall computation time by 68% over iSAM 2.0.

Baltimore, MD, USA, 1996. ISBN 0-8018-5414-8.

[10] J.E. Guivant and E.M. Nebot. Optimization of the simultaneous localization and map-building algorithm for real-time implementation. *Robotics and Automation, IEEE Transactions on*, 17(3):242–257, Jun 2001. ISSN 1042-296X. doi: 10.1109/70.938382.

[11] Pinar Heggernes and Pontus Matstoms. Finding good column orderings for sparse qr factorization. Technical report, In Second SIAM Conference on Sparse Matrices, 1996.

[12] M. Kaess, A. Ranganathan, and F. Dellaert. isam: Incremental smoothing and mapping. *Robotics, IEEE Transactions on*, 24(6):1365–1378, Dec 2008. ISSN 1552-3098. doi: 10.1109/TRO.2008.2006706.

[13] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J.J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *Intl. J. of Robotics Research, IJRR*, 31(2):217–236, Feb 2012.

[14] Michael Kaess, Viorela Ila, Richard Roberts, and Frank Dellaert. The bayes tree: An algorithmic foundation for probabilistic robot mapping. In *Algorithmic Foundations of Robotics IX*, pages 157–173. Springer, 2011.

[15] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2):498–519, Feb 2001. ISSN 0018-9448. doi: 10.1109/18.910572.

[16] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI, 2002.

[17] A. Ranganathan, Ming-Hsuan Yang, and J. Ho. Online sparse gaussian process regression and its applications. *Image Processing, IEEE Transactions on*, 20(2):391–404, Feb 2011. ISSN 1057-7149. doi: 10.1109/TIP.2010.2066984.

[18] C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*. 2006.

[19] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623.

[20] Chi Hay Tong, Paul Furgale, and Timothy D Barfoot. Gaussian process gauss–newton for non-parametric simultaneous localization and mapping. *The International Journal of Robotics Research*, 32(5):507–525, 2013.

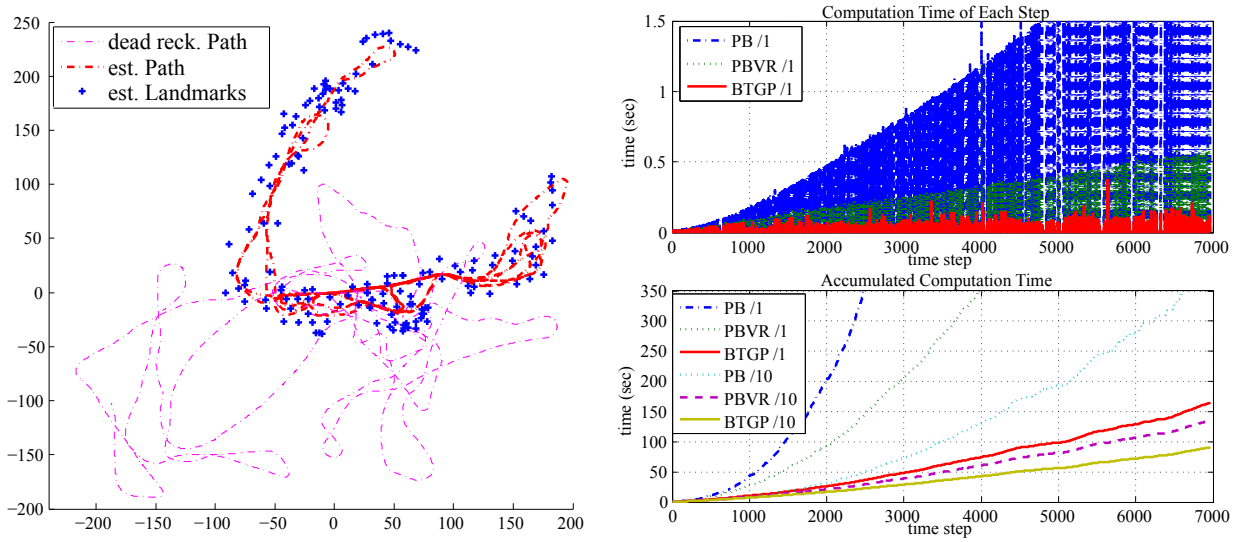


Fig. 8: Victoria Park dataset: (left) dead reckoning and estimated path obtained from BTGP approach. (right) Comparison of the computation time of three approaches PB, PBVR, and BTGP. As in Figures 5 and 7, the modifiers /1 and /10 indicate frequency of state updates. Updates are very fast (close to zero time) when there are no range measurements. Since many landmarks are involved, PBVR dramatically improves performance, compared PB. The incremental BTGP algorithm improves performance even further. Unlike in previous datasets, we did not evaluate the trade-off between interpolation and accuracy for Victoria Park, since we do not have access to ground truth and cannot evaluate the effect on accuracy. However, like previous datasets, interpolation can greatly increase the speed of BTGP.

[21] M. Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981. doi: 10.1137/0602010. URL http://dx.doi.org/10.1137/0602010.